Python

This class uses Python 2.6. (You may also use Python 2.4, but do not use Python 3.) If you're not familiar with Python, there are numerious resources available on the Internet:

- Python tutorial: `http://docs.python.org/tutorial/`
- Python libraries: `http://docs.python.org/library/`
- More Python resources on the 6.006 resources page.

Asymptotic analysis

Asymptotic analysis or "big O" notation is a way of describing the growth of the runtime of an algorithm without without having to worry about different computers, compilers, or implementations.

For functions $f(n)$, $g(n)$, $O(g(n))$ is a class of functions such that $f(n) \in O(g(n))$ if there exist M, x_0 such that

$$|f(n)| \leq M \cdot |g(n)| \text{ for all } x > x_0.$$

Similarly, $f(n) \in \Omega(g(n))$ if there exist M, x_0 such that

$$|f(n)| \geq M \cdot |g(n)| \text{ for all} x > x_0.$$

If $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$, then we write $f(n) \in \Theta(g(n))$.

Less commonly, you'll also see the notation $f(n) \in o(g(n))$ and $f(n) \in \omega(g(n))$. These are sort-of strict versions of O and Ω. More formally, $f(n) \in o(g(n))$ if $\lim_{x\to\infty} \frac{f(n)}{g(n)} = 0$ and $f(n) \in \omega(g(n))$ if $\lim_{x\to\infty} \frac{g(n)}{f(n)} = 0$.

Loosely, we can think of o, O, Θ, Ω, ω as being like $<$, $\leq$, $=$, $\geq$, $>$, but only up to a constant and only for large enough n.

Note: although $O(g(n))$ is really a *class* of functions, we will often write "$f(n) = O(g(n))$" when we mean "$f(n) \in O(g(n))$".

Asymptotic complexity of Python operations

You'll be writing programs in Python and you should be able to predict the asymptotic complexity (runtime) of a Python program. Unfortunately, Python does not specify the complexity of its primitive operations. You can, however, find the actual runtimes on the primitive Python operations at the Python Cost Model webpage (linked to from the Resources page).